Building a Parallel Cloud Storage System using OpenStack's Swift Object Store and Transformative Parallel I/O

Kaleb Lora, Univeristy of Washington Seattle Andrew "AJ" Burns, New Mexico Institute of Mining and Technology
Martel Shorter, Prairie View A&M Esteban Martinez, University of New Mexico

Mentors: HB Chen, HPC-5 Benjamin McCleland, HPC-5 Parks Fields, HPC-5 David Sherrill, HPC-5 Alfred Torrez, HPC-1 Adam Manzanares, HPC-5 Pamela Smith, HPC-3

Instructor: Dane Gardner

Introduction/abstract and objective

In this project we built a prototype of a parallel cloud storage system. We started by determining the scaling capability of Swift's object storage system coupled with the parallel I/O feature from Los Alamos National Laboratories (LANL) Parallel Log-based File System (PLFS).

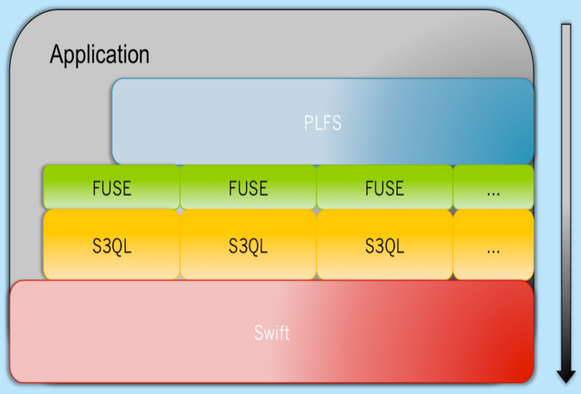
PLFS is used to parallelize and write data in an N-1-N fashion. We wanted to see if and how PLFS would be incorporated in a cloud environment.

Swift is an open source cloud storage application used for creating redundant, scalable object storage using clusters of standardized servers. Swift can store petabytes of accessible data, and it serves as a long-term storage system for more permanent, static data. The data can be retrieved, leveraged, and then updated if necessary.

S3QL is an active Python/FUSE-based file system that runs with Amazon's "Simple Storage Service (S3)". S3 can be transposed into a full-featured UNIX file system which is usable by PLFS. This then serves as the file system interface between LANL's PLFS and OpenStack's Swift object store.

We installed PLFS on top of Swift, and S3QL. After successfully integrating the PLFS transformative parallel I/O feature with our S3QL file systems, we conducted various studies. Typically, we focused on the I/O bandwidth and system performance.

Software Architecture for the proposed Parallel Cloud (Archive) Storage System



Testbed and what tests we ran

The head node was only used as a gateway, and was not involved in any testing. Our compute nodes each had 8 core CPUs and 12GB of RAM. We netbooted all of our compute nodes via Warewulf, with Swift, S3QL, and FUSE instances installed.

Baseline Performance Testing --

Baseline tests began by simply writing to one S3QL mount on a single node using "dd". It wrote 1GB to 4GB test files to S3QL mount with each iteration using a unique block size and count. In terms of speed, we found the optimal block size to be 512KB. As expected, the read tests were much faster.

The second tests involved writing in parallel to 3 local S3QL mounts; again, 512KB was the optimal block size. In this instance, the "dd" command was used again.

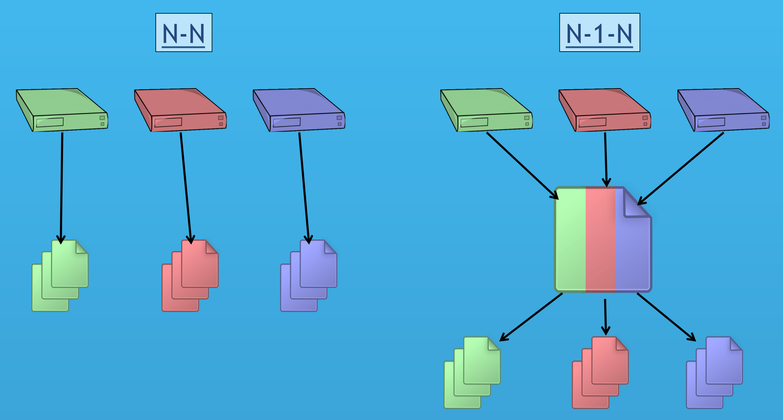
Finally the third baseline test did implement PLFS with double FUSE layers. Having FUSE proves to affect our performance later on.

Target Performance Testing --

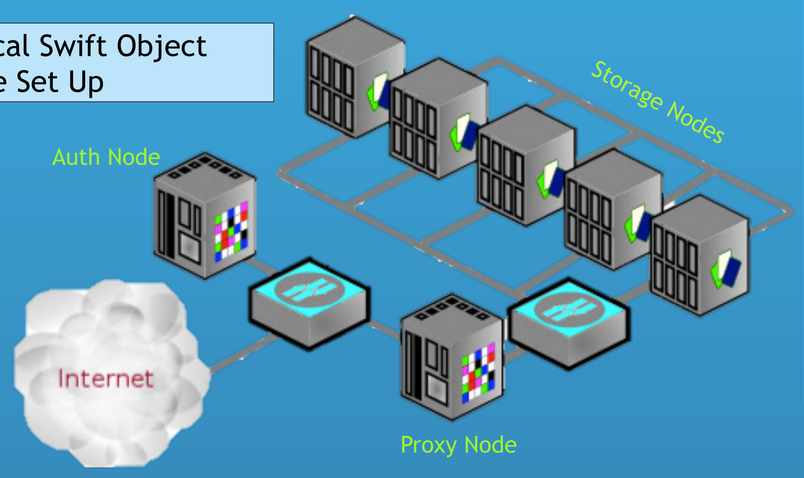
The main goal of our testing was to get to doing writes to many cloud storage objects in parallel. During our target testing we achieved this goal through N-N MPI runs, and PLFS N-1-N write implementation.

We tested the parallel performance of one node, and up to 5 nodes by striping the writes. Each of our nodes had 8 core CPUs so our max performance tests ran 40 processes in parallel to get the best aggregate performance.

Data Writing Methods

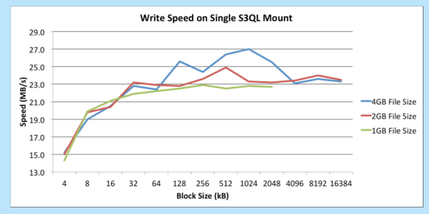


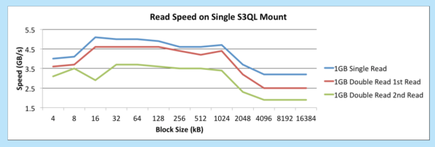
Typical Swift Object Store Set Up



Baseline Performance Tests

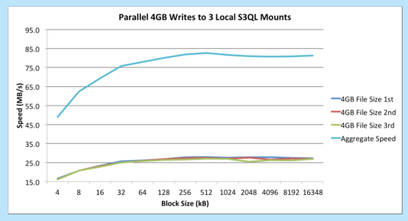
(right) Our first test was done to a single S3QL mount point to determine the best block size. From the graph, we determined 512kB was the best block size to use.

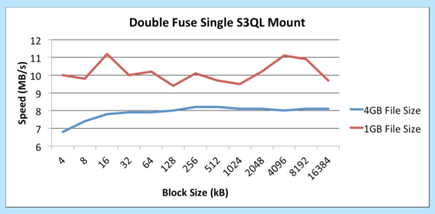




(left) Out of the same test we took read data as well. It isn't overall that interesting due to caching.

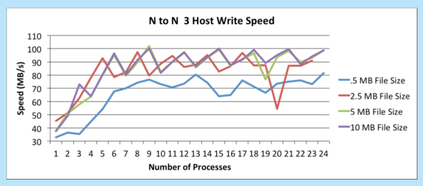
(right) For our next test, we parallelized the writing across 3 S3QL mount points. We got an aggregate speed 3 times higher than a single mount. This shows good single node scaling capabilities.



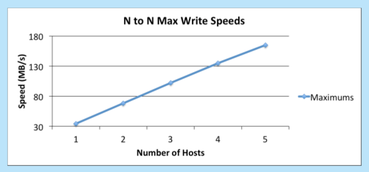


(left) The last baseline test we did involved mounting PLFS through an extra fuse layer. Through this, we found FUSE to be a big limiter in our performance.

Target Performance Tests

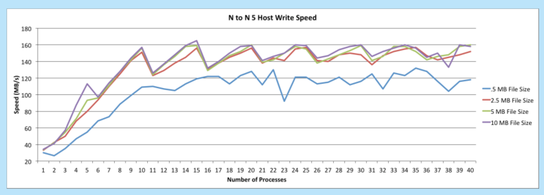


(left) Also noticed spikes of increased performance at each number of processes that was a multiple of the number of hosts we were using.

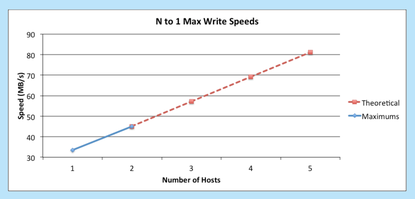


(left) Parallel cloud storage is possible and has good scalability in the N-N case. Performance scaled linearly as nodes were added to our cluster.

(right) Immediate performance improvement with adding nodes even with a small number of processors per node.



(right) PLFS performance results were similar to N-N performance results, but added enough instability to the S3QL mounts that many failures prevented a complete set of tests.



Conclusion

As High Performance Computing moves into the exascale levels, archiving petabytes of data will become a real issue. It is not a viable solution just to buy more tape and hard drives for storage; it simply isn't feasible. Cloud technology is primarily used for reading files, not writing them. Since archiving is about 90% writing files, we needed to determine whether cloud technology could get the performance needed for efficiently writing data. Our research was focused on using cloud storage as a replacement for the traditional methods of archiving data.

The results were what we hoped for: Cloud Storage is a viable archive solution due to its linear scalability as you add hosts to the cloud. Even with the local writes, you get good scalability as you parallelize your writes to the cloud storage file systems. Container management for larger parallel archives eases the migration workloads. Many cloud storage software tools can be utilized for local archives, and with parallelization techniques, bandwidth of archive writes can be dramatically improved.

Future Work

Our current project only focused on writing compressed data to our cloud. There are many interesting things that might be discovered if we tested with different uncompressed data configurations. We would like to examine how the CPU temperature, power consumption, and input bandwidth is affected when writing cached and/or uncompressed data versus the default LZMA compressed data.

As mentioned above in the target testing section, when implementing PLFS, we saw many mount instabilities arise both in our NFS mounts and our S3QL mounts to the Swift cloud server. Future research needs to be done to see if it is possible to improve Swift's stability as it interacts with PLFS by modifying parameters that would allow for scaled, concurrent writes to cloud storage.

Our project only focused on using an S3QL file system, but there are a couple of other viable options that could be tested with Swift's cloud technology. GlusterFS and Ceph are two different file systems that might offer better performance or stability. It would require some testing to determine which type of file system software offers the best performance when interacting with cloud technology.

LA-UR-12-23630